Nominal Anti-Unification

Alexander Baumgartner[1], Temur Kutsia[1], Jordi Levy[2] and Mateu Villaret[3]

[1] Research Institute for Symbolic Computation, Johannes Kepler University Linz, Austria
[2] Artificial Intelligence Research Institute, Spanish Council for Scientific Research (IIIA-CSIC), Barcelona, Spain
[3] Departament d'Informàtica i Matemàtica Aplicada, Universitat de Girona, Spain

1 Introduction

Equation solving between nominal terms has been investigated by several authors, who designed and analyzed algorithms for nominal unification [3, 4, 15, 26], nominal matching [5], equivariant unification [6], permissive nominal unification [8]. However, in contrast to unification, its dual problem, anti-unification, has not been studied for nominal terms previously. In [18], it is referred to as "the as-of-yet undiscovered nominal anti-unification", which "could form a fundamental component of a refactoring tool" for αProlog [7] programs.

Software refactoring is one of possible applications of anti-unification. This method, formulated for different theories, has been successfully used in inductive logic programming [17], cognitive modeling [24], analogy making [13], inductive program synthesis [12], proof generalization [27], mathematical reasoning [10, 11], etc. Nominal anti-unification can play a role in extending some of these applications to the nominal setting. For instance, it can be useful to generalize proofs done in nominal logic, or in doing analogical reasoning in mathematics, or in adapting inductive program synthesis methods to αProlog programs, etc.

The anti-unification problem for two terms t_1 and t_2 is concerned with finding a generalization term t such that t_1 and t_2 are substitutive instances of t. The interesting generalizations are the least general ones (lgg). Plotkin [20] and Reynolds [22] initiated research on anti-unification in the 1970s, developing algorithms for first-order terms. Since then, anti-unification has been studied in various theories, including some of those with binding constructs: calculus of constructions [19], $M\lambda$ [9], second-order lambda calculus with type variables [16], simply-typed lambda calculus where generalizations are higher-order patterns [2], just to name a few.

In this paper we address the problem of computing lgg's for nominal *terms-in-context*, which are pairs of a freshness context and a nominal term. It turned out that without a restriction, there is no lgg for terms-in-context, in general. Therefore we restrict the set of atoms which are permitted in generalizations to be finite. In this case, there exists a single lgg (modulo $\simeq$) and we design an algorithm to compute it. Computation of nominal lgg's requires a solution to the equivariance problem which aims at finding a permutation of atoms π for given terms t_1 and t_2 such that π applied to t_1 is α-equivalent to t_2 (under a given freshness context).

Various anti-unification techniques, such as first-order, higher-order, or equational anti-unification have been used in inductive logic programming, logical and relational learning [21], reasoning by analogy [13], program synthesis [23], program verification [16], etc. Nominal anti-unification can, hopefully, contribute in solving similar problems in nominal setting.

The anti-unification algorithm has been implemented and is available from `www.risc.jku.at/projects/stout/software/nau.php`. The implementation of the equivariance algorithm is also accessible separately from `www.risc.jku.at/projects/stout/software/nequiv.php`.

2 Nominal Terms

Nominal terms contain *variables* and *atoms*. Variables can be instantiated and atoms can be bound. We have *sorts of atoms* ν and *sorts of data* δ as disjoint sets. *Atoms* $(a, b, \ldots)$ have one of the sorts of atoms. *Variables* $(X, Y, \ldots)$ have a sort of atom or data. Nominal function symbols $(f, g, \ldots)$ have an arity of the form $\tau_1 \times \cdots \times \tau_n \to \delta$, where δ is a sort of data and τ_i are sorts given by the grammar $\tau ::= \nu \mid \delta \mid \langle\nu\rangle\tau$. Abstractions have sorts of the form $\langle\nu\rangle\tau$.

A *swapping* $(a\,b)$ is a pair of atoms of the same sort. A *permutation* is a sequence of swappings. We use π, ρ to denote permutations. *Nominal terms* (t, s, r) are given by the grammar:

$$t ::= f(t_1, \ldots, t_n) \mid a \mid a.t \mid \pi{\cdot}X$$

The effect of swapping, and permutation application are defined in the standard way. The *inverse* of a permutation $\pi = (a_1\,b_1)\ldots(a_n\,b_n)$ is the permutation $(a_n\,b_n)\ldots(a_1\,b_1)$, denoted by π^{-1}. We use Id for the empty permutation and write X as the shortcut of $Id{\cdot}X$.

For a set A, we denote by $|A|$ its cardinality. The set of *atoms* of a term t or a permutation π is the set of all atoms which appear in it and is denoted by $\mathrm{Atoms}(t)$, $\mathrm{Atoms}(\pi)$ respectively. $\|t\|_{\mathsf{Abs}}$ stand for the number of abstraction occurrences in t.

Suspensions are uses of variables with a permutation of atoms waiting to be applied once a variable is instantiated. Occurrences of an atom a are said to be bound if they are in the scope of an abstraction of a, otherwise are said to be free. We denote by $\mathrm{FA}(t)$ the set of all atoms which occur freely in t: $\mathrm{FA}(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} \mathrm{FA}(t_i)$, $\mathrm{FA}(a) = \{a\}$, $\mathrm{FA}(a.t) = \mathrm{FA}(t) \setminus \{a\}$, and $\mathrm{FA}(\pi{\cdot}X) = \mathrm{Atoms}(\pi)$. $\mathrm{FA}^{\text{-s}}(t)$ is the set of all atoms which occur freely in t ignoring suspensions: $\mathrm{FA}^{\text{-s}}(f(t_1, \ldots, t_n))$, $\mathrm{FA}^{\text{-s}}(a)$, $\mathrm{FA}^{\text{-s}}(a.t)$ are defined like above but $\mathrm{FA}^{\text{-s}}(\pi{\cdot}X) = \emptyset$.

Substitutions, denoted by σ, are defined in the standard way, and their application allows atom capture, for instance, $a.X\{X \mapsto a\} = a.a$. The identity substitution is denoted by ε.

A *freshness constraint* is a pair of the form $a\#X$ stating that the instantiation of X cannot contain free occurrences of a. A *freshness context* is a finite set of freshness constraints. We will use ∇ and Γ for freshness contexts. $\mathrm{Atoms}(\nabla)$ denotes the set of atoms of ∇.

We say that a substitution σ *respects* ∇, if for all X, $\mathrm{FA}^{\text{-s}}(X\sigma) \cap \{a \mid a\#X \in \nabla\} = \emptyset$.

The predicate $\approx$ stands for α-equivalence and was defined in [25,26] by the following theory:

$$\frac{}{\nabla \vdash a \approx a} \qquad \frac{\nabla \vdash t \approx t'}{\nabla \vdash a.t \approx a.t'} \qquad \frac{a \neq a' \quad \nabla \vdash t \approx (a\,a'){\cdot}t' \quad \nabla \vdash a\#t'}{\nabla \vdash a.t \approx a'.t'}$$

$$\frac{a\#X \in \nabla \text{ for all } a \text{ such that } \pi{\cdot}a \neq \pi'{\cdot}a}{\nabla \vdash \pi{\cdot}X \approx \pi'{\cdot}X} \qquad \frac{\nabla \vdash t_1 \approx t'_1 \quad \cdots \quad \nabla \vdash t_n \approx t'_n}{\nabla \vdash f(t_1, \ldots t_n) \approx f(t'_1, \ldots, t'_n)}$$

where the freshness predicate $\#$ is defined by

$$\frac{a \neq a'}{\nabla \vdash a\#a'} \qquad \frac{(\pi^{-1}{\cdot}a\#X) \in \nabla}{\nabla \vdash a\#\pi{\cdot}X} \qquad \frac{\nabla \vdash a\#t_1 \quad \cdots \quad \nabla \vdash a\#t_n}{\nabla \vdash a\#f(t_1, \ldots t_n)} \qquad \frac{}{\nabla \vdash a\#a.t} \qquad \frac{a \neq a' \quad \nabla \vdash a\#t}{\nabla \vdash a\#a'.t}$$

Given a freshness context ∇ and a substitution σ, we define $\nabla\sigma$ as the *minimal* (with respect to $\subseteq$) freshness context such that for all $a\#X \in \nabla$ holds $\nabla\sigma \vdash a\#X\sigma$. It can easily be derived from the definition of the freshness predicate, if it exists. Otherwise it is undefined.

Theorem 1. *σ respects ∇ iff $\nabla\sigma$ is defined.*

A *term-in-context*, denoted by p, is a pair $\langle\nabla, t\rangle$ of a freshness context and a term. $\langle\nabla, t\rangle$ is *more general* than a term-in-context $\langle\Gamma, s\rangle$, written $\langle\nabla, t\rangle \preceq \langle\Gamma, s\rangle$, if there is a substitution σ, which respects ∇, such that $\nabla\sigma \subseteq \Gamma$ and $\Gamma \vdash t\sigma \approx s$. We write $\nabla \vdash t \preceq s$ if there exists a substitution σ such that $\nabla \vdash t\sigma \approx s$. We also write $\nabla \vdash t \simeq s$ iff $\nabla \vdash t \preceq s$ and $\nabla \vdash s \preceq t$.

Example 1. We give some examples to demonstrate the relations we have just defined:

- $\langle\{a\#X\}, f(a)\rangle \simeq \langle\emptyset, f(a)\rangle$. We can use $\{X \mapsto b\}$ as substitution applied to the first pair.
- $\langle\emptyset, f(X)\rangle \preceq \langle\{a\#Y\}, f(Y)\rangle$ with $\sigma = \{X \mapsto Y\}$, but not $\langle\{a\#Y\}, f(Y)\rangle \preceq \langle\emptyset, f(X)\rangle$.
- $\langle\{a\#X\}, f(X)\rangle \not\preceq \langle\{a\#X\}, f(a)\rangle$. Notice that $\sigma = \{X \mapsto a\}$ does not respect $\{a\#X\}$.
- $\langle\{b\#X\}, (a\,b)\cdot X\rangle \preceq \langle\{c\#X\}, (a\,c)\cdot X\rangle$ with the substitution $\sigma = \{X \mapsto (a\,b)(a\,c)\cdot X\}$.

A term-in-context $\langle\Gamma, r\rangle$ is called a *generalization* of two terms-in-context $\langle\nabla_1, t\rangle$ and $\langle\nabla_2, s\rangle$ if $\langle\Gamma, r\rangle \preceq \langle\nabla_1, t\rangle$ and $\langle\Gamma, r\rangle \preceq \langle\nabla_2, s\rangle$. It is the *least general generalization* (lgg) of $\langle\nabla_1, t\rangle$ and $\langle\nabla_2, s\rangle$ if there is no generalization $\langle\Gamma', r'\rangle$ of $\langle\nabla_1, t\rangle$ and $\langle\nabla_2, s\rangle$ which satisfies $\langle\Gamma, r\rangle \prec \langle\Gamma', r'\rangle$.

Note that if we have infinite number of atoms in the language, the relation $\prec$ is not well-founded: $\langle\emptyset, X\rangle \prec \langle\{a\#X\}, X\rangle \prec \langle\{a\#X, b\#X\}, X\rangle \prec \cdots$. As a consequence, two terms-in-context may not have an lgg and not even a minimal complete set of generalizations:[1]

Example 2. Let $p_1 = \langle\emptyset, a_1\rangle$ and $p_2 = \langle\emptyset, a_2\rangle$ be two terms-in-context. Then in any complete set of generalizations of p_1 and p_2 there is an infinite chain $\langle\emptyset, X\rangle \prec \langle\{a_3\#X\}, X\rangle \prec \langle\{a_3\#X, a_4\#X\}, X\rangle \prec \cdots$, where $\{a_1, a_2, a_3, \ldots\}$ is the set of all atoms of the language. Hence, p_1 and p_2 do not have a minimal complete set of generalizations.

Theorem 2. *The problem of anti-unification for terms-in-context is of nullary type.*

However, if we restrict the set of atoms which can be used in the generalizations to be finite, then the anti-unification problem becomes unitary.

We say that a term t (resp., a freshness context ∇) is *based* on a set of atoms A iff $\mathrm{Atoms}(t) \subseteq A$ (resp., $\mathrm{Atoms}(\nabla) \subseteq A$). A term-in-context $\langle\nabla, t\rangle$ is based on A if both t and ∇ are based on it. A permutation is A-based if it contains only atoms from A. An *A-based lgg* of A-based terms-in-context p_1 and p_2 is an A-based term-in-context p, which is a generalization of p_1 and p_2 and there is no A-based generalization p' of p_1 and p_2 which satisfies $p \prec p'$.

3 Nominal Anti-Unification Algorithm

Our anti-unification problem is parametric on the set of atoms we consider as the base, and finiteness of this set is essential to ensure the existence of an lgg. The problem we would like to solve is the following:

Given: Two nominal terms t and s of the same sort, a freshness context ∇, and a *finite* set of atoms A such that t, s, and ∇ are based on A.

Find: A term r and a freshness context Γ, such that the term-in-context $\langle\Gamma, r\rangle$ is an A-based least general generalization of the terms-in-context $\langle\nabla, t\rangle$ and $\langle\nabla, s\rangle$.

The triple $X : t \triangleq s$, where X, t, s have the same sort, is called the *anti-unification equation*, shortly AUE, and the variable X is called a *generalization variable*. We say that a set of AUEs P is based on a finite set of atoms A, if for all $X : t \triangleq s \in P$, the terms t and s are A-based.

The anti-unification algorithm is formulated in a rule-based way and depends on two global parameters, a finite set of atoms A and a freshness context ∇. It works on tuples of the form P; S; Γ; σ, where P and S are sets of AUEs, Γ is a freshness context and σ is a substitution. P, S, ∇, and Γ are A-based and ∇ does not constrain generalization variables. Furthermore if $X : t \triangleq s \in P \cup S$, then this is the sole occurrence of X in $P \cup S$. The rules are the following:

[1] The definition of minimal complete sets of generalizations is standard. For a precise definition, see, e.g. [1,14].

Dec: **Decomposition**

$\{X : \mathsf{h}(t_1, \ldots, t_m) \triangleq \mathsf{h}(s_1, \ldots, s_m)\} \uplus P;\ S;\ \Gamma;\ \sigma \Longrightarrow$
$\{Y_1 : t_1 \triangleq s_1, \ldots, Y_m : t_m \triangleq s_m\} \cup P;\ S;\ \Gamma;\ \sigma\{X \mapsto \mathsf{h}(Y_1, \ldots, Y_m)\}$,

where h is a function symbol or an atom, $Y_1, \ldots, Y_m$ are fresh variables of appropriate sorts.

Abs: **Abstraction**

$\{X : a.t \triangleq b.s\} \uplus P;\ S;\ \Gamma;\ \sigma \Longrightarrow \{Y : (c\ a)\cdot t \triangleq (c\ b)\cdot s\} \cup P;\ S;\ \Gamma;\ \sigma\{X \mapsto c.Y\}$,

where Y is fresh, $c \in A$ such that $\nabla \vdash c\#a.t$ and $\nabla \vdash c\#b.s$.

Sol: **Solving**

$\{X : t \triangleq s\} \uplus P;\ S;\ \Gamma;\ \sigma \Longrightarrow P;\ S \cup \{X : t \triangleq s\};\ \Gamma \cup \Gamma';\ \sigma$,

if neither Dec nor Abs is applicable, where $\Gamma' = \{a\#X \mid a \in A\ \wedge\ \nabla \vdash a\#t\ \wedge\ \nabla \vdash a\#s\}$.

Mer: **Merging**

$P;\ \{X : t_1 \triangleq s_1, Y : t_2 \triangleq s_2\} \uplus S;\ \Gamma;\ \sigma \Longrightarrow$
$P;\ \{X : t_1 \triangleq s_1\} \cup S;\ \Gamma\{Y \mapsto \pi\cdot X\};\ \sigma\{Y \mapsto \pi\cdot X\}$,

where π is an A-based permutation such that $\nabla \vdash \pi\cdot t_1 \approx t_2$, and $\nabla \vdash \pi\cdot s_1 \approx s_2$.

Given a finite set of atoms A, an A-based freshness context ∇, and two nominal A-based terms t and s, to compute an A-based generalization for $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$, we start with $\{X : t \triangleq s\}; \emptyset; \emptyset; \varepsilon$, where X is a fresh variable, and apply the rules (don't care) nondeterministically as long as possible. We denote this procedure by $\mathcal{N}$, and say that the *final state* is reached when no more rule is applicable. The final state is of the form $\emptyset; S; \Gamma; \sigma$, where Mer does not apply to S and we say that the *result computed* by $\mathcal{N}$ is $\langle \Gamma, X\sigma\rangle$.

Note that the Dec rule works also for the AUEs of the form $X : a \triangleq a$. In the Abs rule, it is important to have the corresponding c in A. If not then the Sol rule takes over.

Example 3. We illustrate $\mathcal{N}$ on a couple of examples:

- Let $t = f(a, b)$, $s = f(b, c)$, $\nabla = \emptyset$, and $A = \{a, b, c, d\}$. Then $\mathcal{N}$ performs the following transformations:

$$\{X : f(a,b) \triangleq f(b,c)\}; \emptyset; \emptyset; \varepsilon \Longrightarrow_{\mathsf{Dec}}$$
$$\{Y : a \triangleq b, Z : b \triangleq c\}; \emptyset; \emptyset; \{X \mapsto f(Y,Z)\} \Longrightarrow^{2}_{\mathsf{Sol}}$$
$$\emptyset; \{Y : a \triangleq b, Z : b \triangleq c\}; \{c\#Y, d\#Y, a\#Z, d\#Z\};\ \{X \mapsto f(Y,Z)\} \Longrightarrow_{\mathsf{Mer}}$$
$$\emptyset; \{Y : a \triangleq b\}; \{c\#Y, d\#Y\}; \{X \mapsto f(Y, (a\,b)(b\,c)\cdot Y)\}$$

  Hence, $p = \langle \{c\#Y, d\#Y\}, f(Y, (a\,b)(b\,c)\cdot Y)\rangle$ is the computed result. It generalizes the input pairs: $p\{Y \mapsto a\} \preceq \langle \nabla, t\rangle$ and $p\{Y \mapsto b\} \preceq \langle \nabla, s\rangle$. The substitutions $\{Y \mapsto a\}$ and $\{Y \mapsto b\}$ can be read from the final store. Note that $\langle \{c\#Y\}, f(Y, (a\,b)(b\,c)\cdot Y)\rangle$ would be also an A-based generalization of $\langle \nabla, t\rangle$ and $\langle \nabla, s\rangle$, but it is strictly more general than p.
- Let $t = f(b, a)$, $s = f(Y, (a\,b)\cdot Y)$, $\nabla = \{b\#Y\}$, and $A = \{a, b\}$. Then $\mathcal{N}$ computes the term-in-context $\langle \emptyset, f(Z_1, (a\,b)\cdot Z_1)\rangle$ which generalizes the input pairs.
- Let $t = f(a.b, X)$, $s = f(b.a, Y)$, $\nabla = \{c\#X\}$, and $A = \{a, b, c, d\}$. Then $\mathcal{N}$ computes the term-in-context $p = \langle \{c\#Z_1, d\#Z_1\}, f(c.Z_1, Z_2)\rangle$ which generalizes the input: $p\{Z_1 \mapsto b, Z_2 \mapsto X\} = \langle \emptyset, f(c.b, X)\rangle \preceq \langle \nabla, t\rangle$ and $p\{Z_1 \mapsto a, Z_2 \mapsto Y\} = \langle \emptyset, f(c.a, Y)\rangle \preceq \langle \nabla, s\rangle$.

Theorem 3. *Let t, s be terms and ∇, Γ be freshness contexts, all based on a finite atoms set A.*

- Termination: *The procedure $\mathcal{N}$ terminates on any input.*

- Soundness: *If* $\{X : t \triangleq s\}; \emptyset; \emptyset; \varepsilon \Longrightarrow^+ \emptyset; S; \Gamma; \sigma$ *is a derivation obtained by an execution of* $\mathcal{N}$*, then* $\langle \Gamma, X\sigma \rangle$ *is an* A*-based generalization of* $\langle \nabla, t \rangle$ *and* $\langle \nabla, s \rangle$.
- Completeness: *If* $\langle \Gamma, r \rangle$ *is an* A*-based generalization of* $\langle \nabla, t \rangle$ *and* $\langle \nabla, s \rangle$*, then there exists a derivation* $\{X : t \triangleq s\}; \emptyset; \emptyset; \varepsilon \Longrightarrow^+ \emptyset; S; \Gamma'; \sigma$ *obtained by an execution of* $\mathcal{N}$*, such that* $\langle \Gamma, r \rangle \preceq \langle \Gamma', X\sigma \rangle$.
- Uniqueness: *Let* $\{X : t \triangleq s\}; \emptyset; \emptyset; \varepsilon \Longrightarrow^+ \emptyset; S_1; \Gamma_1; \sigma_1$ *and* $\{X : t \triangleq s\}; \emptyset; \emptyset; \varepsilon \Longrightarrow^+ \emptyset; S_2; \Gamma_2; \sigma_2$ *be two maximal derivations in* $\mathcal{N}$*. Then* $\langle \Gamma_1, X\sigma_1 \rangle \simeq \langle \Gamma_2, X\sigma_2 \rangle$.

Now we study how lgg's of terms-in-context depend on the set of atoms they are based on.

Lemma 1. *Let* A_1 *and* A_2 *be two finite sets of atoms with* $A_1 \subseteq A_2$ *such that the* A_1*-based terms-in-context* $\langle \nabla, t \rangle$ *and* $\langle \nabla, s \rangle$ *have an* A_1*-based lgg* $\langle \Gamma_1, r_1 \rangle$ *and an* A_2*-based lgg* $\langle \Gamma_2, r_2 \rangle$*. Then* $\Gamma_2 \vdash r_1 \preceq r_2$.

In general, we can not replace $\Gamma_2 \vdash r_1 \preceq r_2$ with $\Gamma_2 \vdash r_1 \simeq r_2$ in Lemma 1. Consider for instance the example $t = a.b$, $s = b.a$, $\nabla = \emptyset$, $A_1 = \{a, b\}$, and $A_2 = \{a, b, c\}$. Then for $\langle \nabla, t \rangle$ and $\langle \nabla, s \rangle$, $\langle \emptyset, X \rangle$ is an A_1-based lgg and $\langle \{c\#X\}, c.X \rangle$ is an A_2-based lgg. Obviously, $\{c\#X\} \vdash X \preceq c.X$ but not $\{c\#X\} \vdash c.X \preceq X$.

We say that a set of atoms A is *saturated* for A-based t, s and ∇, if

$$|A \setminus (\mathrm{Atoms}(t) \cup \mathrm{Atoms}(s) \cup \mathrm{Atoms}(\nabla))| \geq \min\{\|t\|_{\mathsf{Abs}}, \|s\|_{\mathsf{Abs}}\}.$$

Lemma 2. *Under the conditions of Lemma 1, if* A_1 *is saturated for* t, s, ∇*, then* $\Gamma_2 \vdash r_1 \simeq r_2$.

4 Deciding Equivariance

Computation of π in the condition of the rule Mer above requires an algorithm that solves the following problem: Given a finite set of atoms A, terms t and s, and a freshness context ∇, all based on A, find an A-based permutation π such that $\nabla \vdash \pi{\cdot}t \approx s$. This is the problem of deciding whether t and s are equivariant with respect to ∇ and A.

We describe a rule-base algorithm, which we call $\mathcal{E}$, that solves this problem by effectively computing the corresponding permutation. It works on tuples of the form $E; \nabla; A; \pi$ (called systems). E is a set of equivariance equations of the form $t \approx s$ where t, s are nominal terms. ∇ is a freshness context, and A is a finite set of atoms which are available for computing π. The latter holds the permutation to be returned in case of success.

The algorithm is split into two phases. In phase 1, function applications, abstractions, and suspensions are decomposed as long as possible. Phase 2 is the permutation computation.

Phase 1 – Dec-E: **Decomposition**

$$\{f(t_1, \ldots, t_m) \approx f(s_1, \ldots, s_m)\} \cup E;\ \nabla;\ A;\ Id \Longrightarrow \{t_1 \approx s_1, \ldots, t_m \approx s_m\} \cup E;\ \nabla;\ A;\ Id.$$

Phase 1 – Alp-E: **Alpha Equivalence**

$$\{a.t \approx b.s\} \cup E;\ \nabla;\ A;\ Id \Longrightarrow \{(a\ c')\cdot t \approx (b\ c')\cdot s\} \cup E;\ \nabla;\ A;\ Id,$$

where c' is a fresh atom of the same sort as a and b.

Phase 1 – Sus-E: **Suspension**

$$\{\pi_1{\cdot}X \approx \pi_2{\cdot}X\} \cup E;\ \nabla;\ A;\ Id \Longrightarrow \{\pi_1{\cdot}a \approx \pi_2{\cdot}a \mid a \in A \wedge a\#X \notin \nabla\} \cup E;\ \nabla;\ A;\ Id.$$

Phase 2 – Rem-E: **Remove**

$$\{a \approx b\} \cup E;\ \nabla;\ A;\ \pi \Longrightarrow E;\ \nabla;\ A \setminus \{b\};\ \pi, \qquad \text{if } \pi{\cdot}a = b.$$

Phase 2 – Sol-E: **Solve**

$$\{a \approx b\} \cup E;\ \nabla;\ A;\ \pi \Longrightarrow E;\ \nabla;\ A \setminus \{b\};\ (\pi{\cdot}a\ b)\pi, \qquad \text{if } \pi{\cdot}a, b \in A \text{ and } \pi{\cdot}a \neq b.$$

The input for $\mathcal{E}$ is initialized in the Mer rule, which needs to compute an A-based permutation π for A-based context ∇ and two AUEs $X : t_1 \triangleq s_1$ and $Y : t_2 \triangleq s_2$. The system is initialized by $\{t_1 \approx t_2, s_1 \approx s_2\}; \nabla; A; Id$. First we apply the rules of phase 1 exhaustively and afterwards Rem-E and Sol-E are applied as long as possible. If the final system is the *success state* $\emptyset; \nabla; A; \pi$, then we say that $\mathcal{E}$ *computes* the permutation π. Otherwise it has the form $E; \nabla; A; \pi$ with $E \neq \emptyset$ to which no rule applies. It is transformed into $\bot$, called the *failure state*.

Example 4. We illustrate the algorithm $\mathcal{E}$ on examples and consider the equivariance problems:

- For $E = \{a \approx a,\ a.(a\,b)(c\,d)\cdot X \approx b.X\}$, $A = \{a, b, c, d\}$, and $\nabla = \{a\#X\}$, we derive
$$\{a \approx a, a.(a\,b)(c\,d)\cdot X \approx b.X\}; \{a\#X\};\ \{a, b, c, d\};\ Id \Longrightarrow_{\text{Alp-E}}$$
$$\{a \approx a, (a\,é)(a\,b)(c\,d)\cdot X \approx (b\,é)\cdot X\}; \{a\#X\};\ \{a, b, c, d\};\ Id \Longrightarrow_{\text{Sus-E}}$$
$$\{a \approx a, é \approx é, c \approx d, d \approx c\}; \{a\#X\};\ \{a, b, c, d\};\ Id \Longrightarrow^{2}_{\text{Rem-E}}$$
$$\{c \approx d, d \approx c\}; \{a\#X\};\ \{b, c, d\};\ Id \Longrightarrow^{\text{Sol-E}}_{\text{Rem-E}} \quad \emptyset;\ \{a\#X\};\ \{b\};\ (c\,d).$$
- For $E = \{a.b.(a\,b)(a\,c)\cdot X = b.a.(a\,c)\cdot X\}$, $A = \{a, b\}$, and $\nabla = \emptyset$, $\mathcal{E}$ returns Id.

Theorem 4. *Let t, s be terms and ∇ be a freshness context, all based on a finite set of atoms A.*

- Termination: *The procedure $\mathcal{E}$ terminates on any input.*
- Soundness: *Let $\{t \approx s\}; \nabla; A; Id \Longrightarrow^* \emptyset; \nabla; B; \pi$ be a derivation in $\mathcal{E}$, then π is an A-based permutation such that $\nabla \vdash \pi\cdot t \approx s$.*
- Completeness: *If $\nabla \vdash \rho\cdot t \approx s$ for some A-based permutation ρ, then there is a derivation $\{t \approx s\}; \nabla; A; Id \Longrightarrow^* \emptyset; \Gamma; B; \pi$, obtained by $\mathcal{E}$, such that $\pi\cdot a = \rho\cdot a$ for all $a \in \mathrm{FA}(t)$.*

Theorem 5. *The equivariance algorithm $\mathcal{E}$ has $O(n^2)$ space and time complexity and the anti-unification algorithm $\mathcal{N}$ has $O(n^4)$ time and $O(n^2)$ space complexity, where n is the input size.*

Acknowledgment

This research has been partially supported by the project HeLo (TIN2012-33042) and by the Austrian Science Fund (FWF) with the project SToUT (P 24087-N18).

References

[1] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. A modular equational generalization algorithm. In M. Hanus, editor, *LOPSTR*, volume 5438 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2008.

[2] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. A variant of higher-order anti-unification. In F. van Raamsdonk, editor, *RTA*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[3] C. Calvès. *Complexity and Implementation of Nominal Algorithms*. PhD thesis, Kings College London, 2010.

[4] C. Calvès and M. Fernández. A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306, 2008.

[5] C. Calvès and M. Fernández. Matching and alpha-equivalence check for nominal terms. *J. Comput. Syst. Sci.*, 76(5):283–301, 2010.

[6] J. Cheney. Equivariant unification. *JAR*, 45(3):267–300, 2010.

[7] J. Cheney and C. Urban. alpha-Prolog: A logic programming language with names, binding and a-equivalence. In B. Demoen and V. Lifschitz, editors, *ICLP*, volume 3132 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2004.

[8] G. Dowek, M. J. Gabbay, and D. P. Mulligan. Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. *Logic Journal of the IGPL*, 18(6):769–822, 2010.

[9] C. Feng and S. Muggleton. Towards inductive generalization in higher order logic. In D. H. Sleeman and P. Edwards, editors, *ML*, pages 154–162. Morgan Kaufmann, 1992.

[10] M. Guhe, A. Pease, A. Smaill, M. Martínez, M. Schmidt, H. Gust, K.-U. Kühnberger, and U. Krumnack. A computational account of conceptual blending in basic mathematics. *Cognitive Systems Research*, 12(3-4):249–265, 2011.

[11] M. Guhe, A. Pease, A. Smaill, M. Schmidt, H. Gust, K.-U. Kühnberger, and U. Krumnack. Mathematical reasoning with higher-order anti-unifcation. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, pages 1992–1997, 2010.

[12] E. Kitzelmann and U. Schmid. Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research*, 7:429–454, 2006.

[13] U. Krumnack, A. Schwering, H. Gust, and K.-U. Kühnberger. Restricted higher-order anti-unification for analogy making. In M. A. Orgun and J. Thornton, editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 273–282. Springer, 2007.

[14] T. Kutsia, J. Levy, and M. Villaret. Anti-unification for unranked terms and hedges. In M. Schmidt-Schauß, editor, *RTA*, volume 10 of *LIPIcs*, pages 219–234. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[15] J. Levy and M. Villaret. Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10, 2012.

[16] J. Lu, J. Mylopoulos, M. Harao, and M. Hagiya. Higher order generalization and its application in program verification. *Ann. Math. Artif. Intell.*, 28(1-4):107–126, 2000.

[17] S. Muggleton. Inverse entailment and progol. *New Generation Comput.*, 13(3&4):245–286, 1995.

[18] D. Mulligan. *Extensions of Nominal Terms*. PhD thesis, School of Math. and Comp. Sci., Heriot-Watt University, Edinburgh, 2011.

[19] F. Pfenning. Unification and anti-unification in the calculus of constructions. In *LICS*, pages 74–85. IEEE Computer Society, 1991.

[20] G. D. Plotkin. A note on inductive generalization. *Machine Intel.*, 5(1):153–163, 1970.

[21] L. D. Raedt. *Logical and Relational Learning*. Springer, 2008.

[22] J. C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intel.*, 5(1):135–151, 1970.

[23] U. Schmid. *Inductive Synthesis of Functional Programs, Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning*, volume 2654 of *Lecture Notes in Computer Science*. Springer, 2003.

[24] A. Schwering, U. Krumnack, K.-U. Kühnberger, and H. Gust. Syntactic principles of heuristic-driven theory projection. *Cognitive Systems Research*, 10(3):251–269, 2009.

[25] C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. In M. Baaz and J. A. Makowsky, editors, *CSL*, volume 2803 of *Lecture Notes in Computer Science*, pages 513–527. Springer, 2003.

[26] C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1–3):473–497, 2004.

[27] C. Walther and T. Kolbe. Proving theorems by reuse. *Artif. Intell.*, 116(1-2):17–66, 2000.